Memfault

Wrangling Penguins: Better Embedded Linux Monitoring and Debugging with Memfault

Thomas Sarlandie, Linux Tech Lead

Thomas Sarlandie

Linux Tech Lead, Memfault

- Passion: building at the intersection of software and hardware
- Previously led software teams at Pebble and Fitbit
- 🦀 🦀 🦀 Rust-aficionado

pebble fitbit Memfault

Agenda

- Monitoring embedded linux devices
- Monitoring a fleet of embedded devices
- Debugging with logs and coredumps
- Q & A

Poll #1

Which of these tools do you use to monitor and debug your fleet in production?

Check all that apply

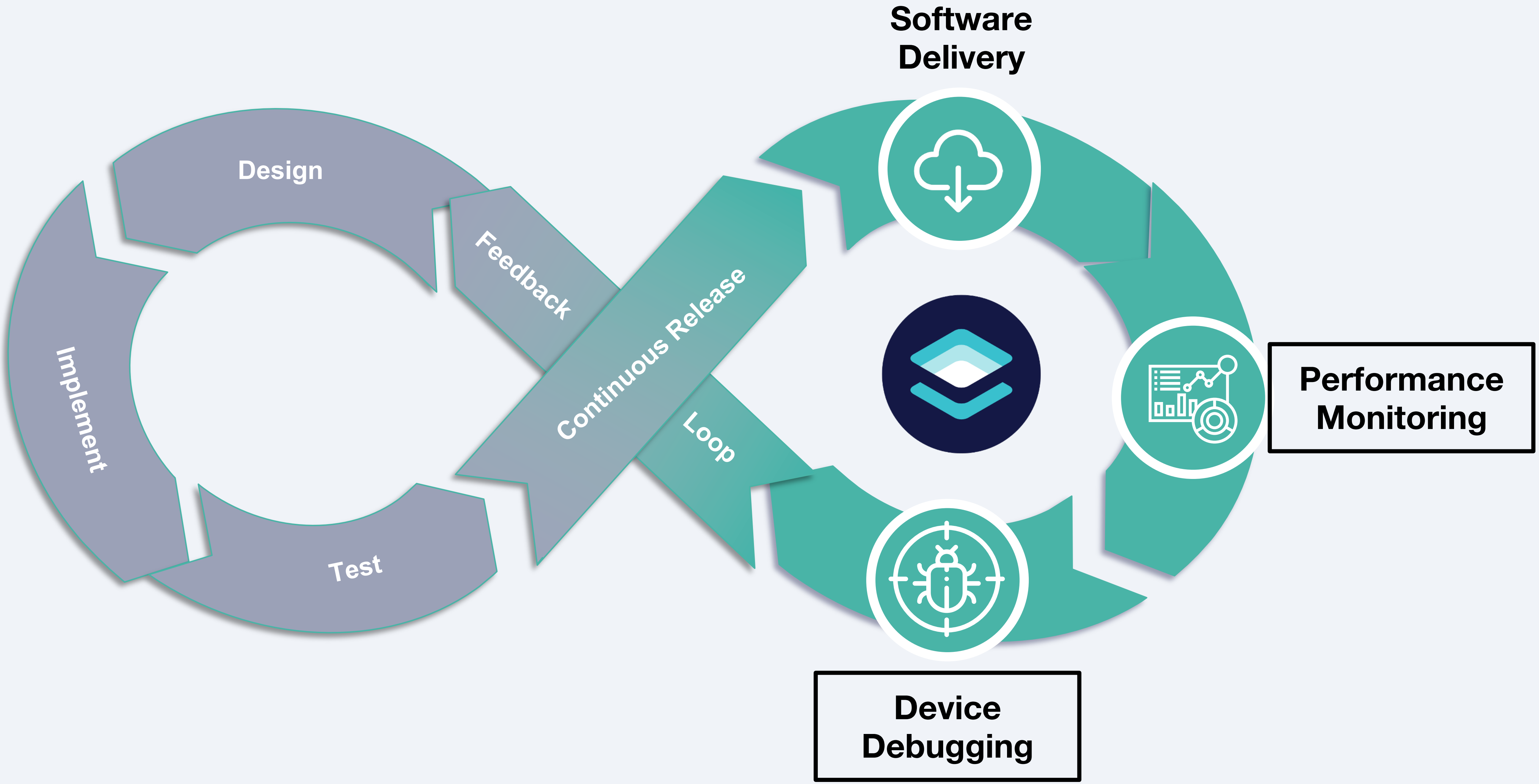
A. SSH

B. Grafana

C. Coredumps

D. Logs

Memfault for Device Reliability Engineering



Monitoring embedded linux devices

Monitoring Goals

Validate hypotheses and debug device issues

Get a pulse on the fleet - especially when shipping new hardware or firmware

Detect problems before the customers

Monitoring Challenges

On Device

- Collecting from different sources and languages
- Partial connectivity
- Flash wear and networking costs

Backend

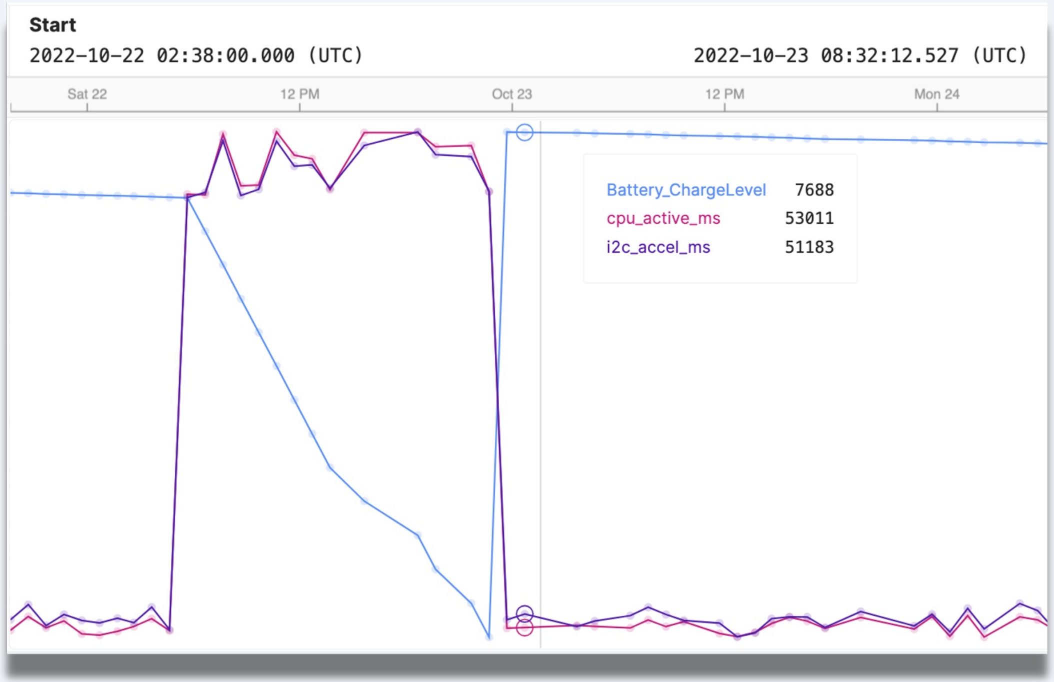
- Scaling pains
- Lack of flexibility
- Visualization tools

Usage

- Drowning in data
- Metrics are meaningless when aggregated
- Signal lost in the data

Metrics

A **metric** is a measurement captured at runtime

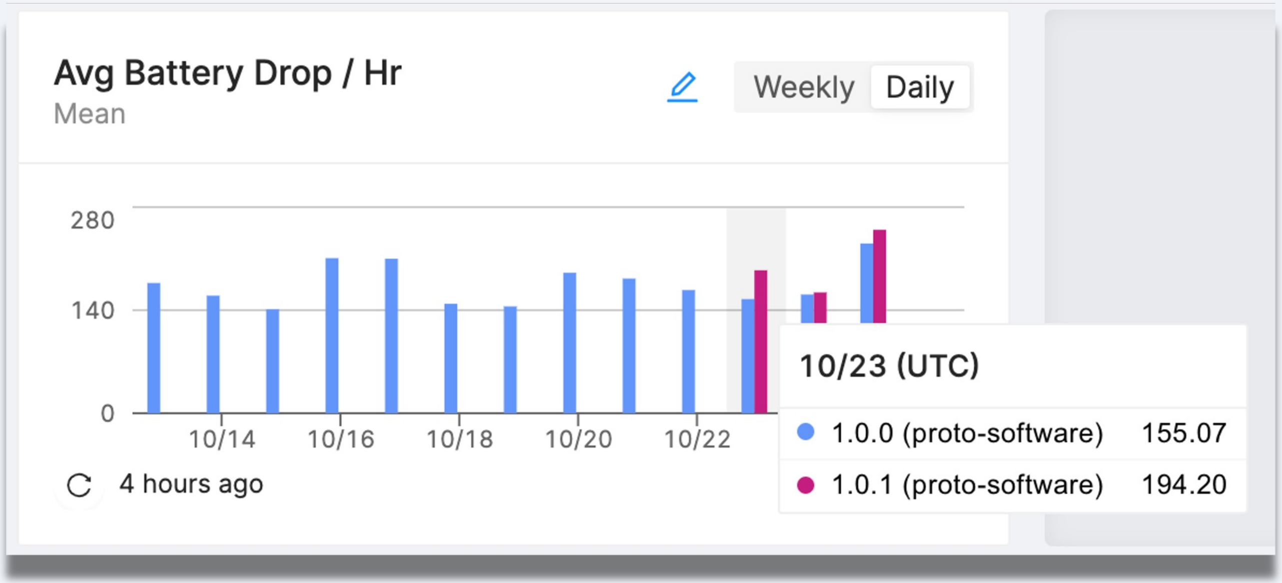


Start

2022-10-22 02:38:00.000 (UTC)

2022-10-23 08:32:12.527 (UTC)

Battery_ChargeLevel	7688
cpu_active_ms	53011
i2c_accel_ms	51183



Avg Battery Drop / Hr

Mean

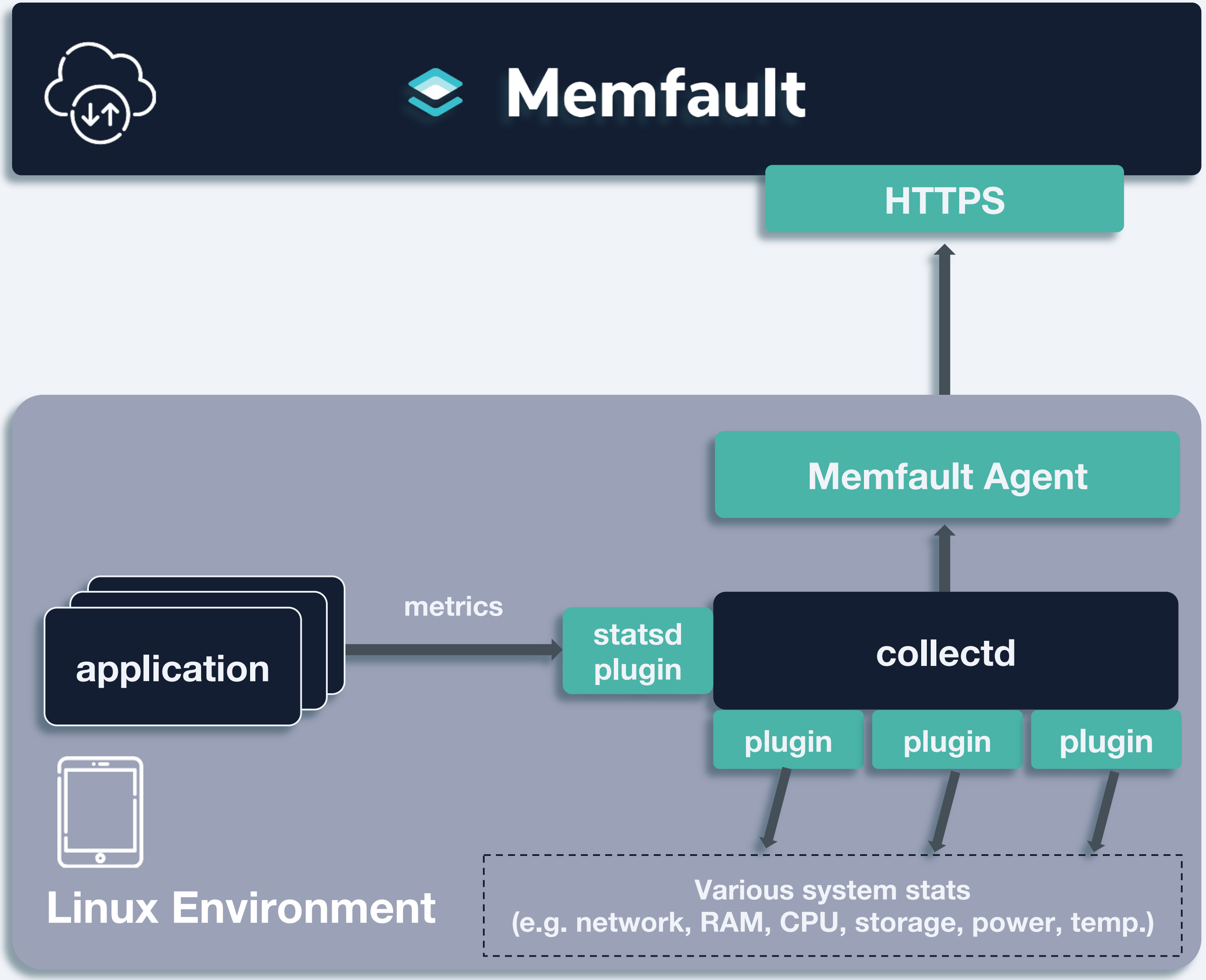
10/23 (UTC)

● 1.0.0 (proto-software)	155.07
● 1.0.1 (proto-software)	194.20

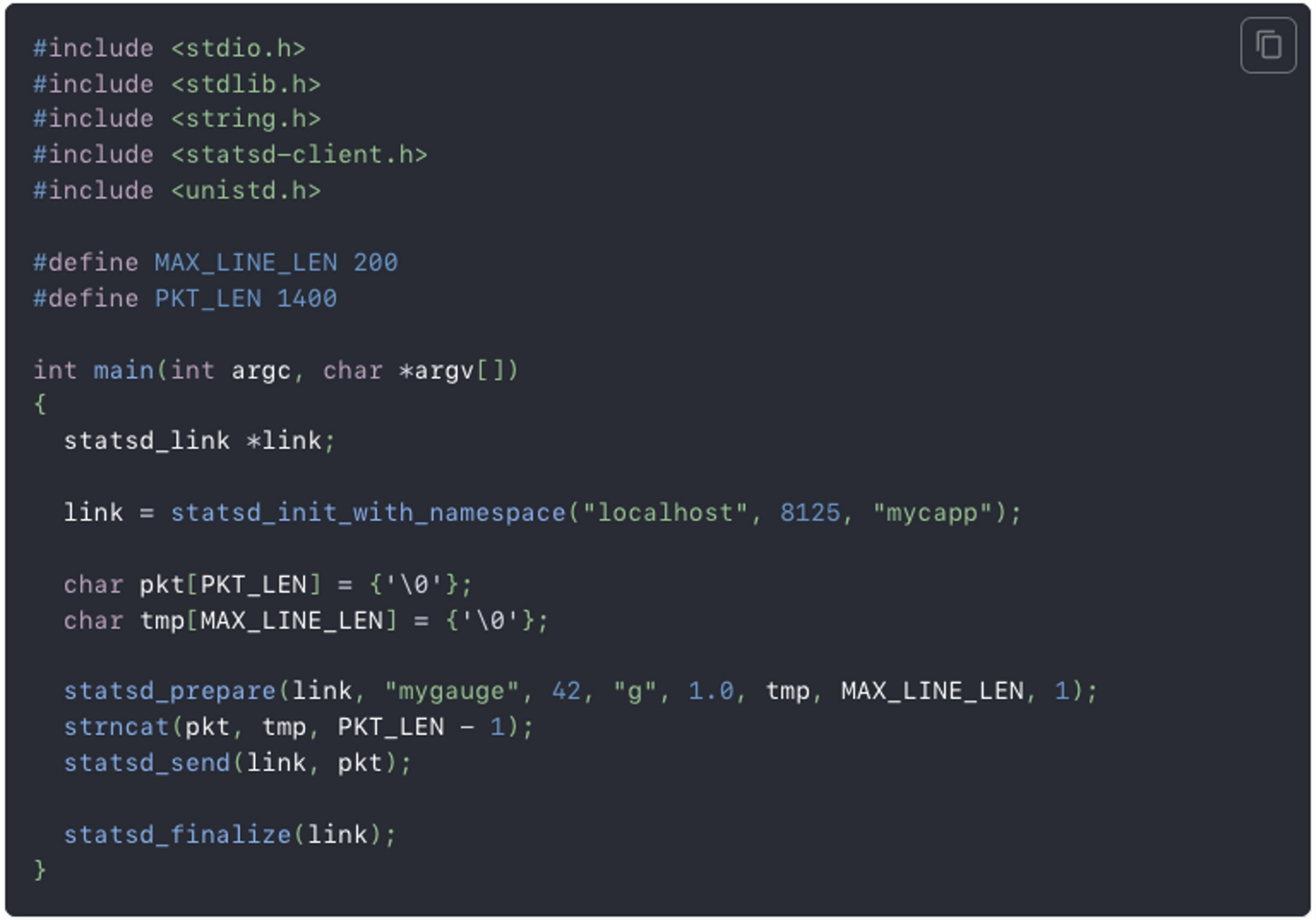
Combing large numbers of metrics and calculating statistics is called an **aggregation**

Collecting metrics on device

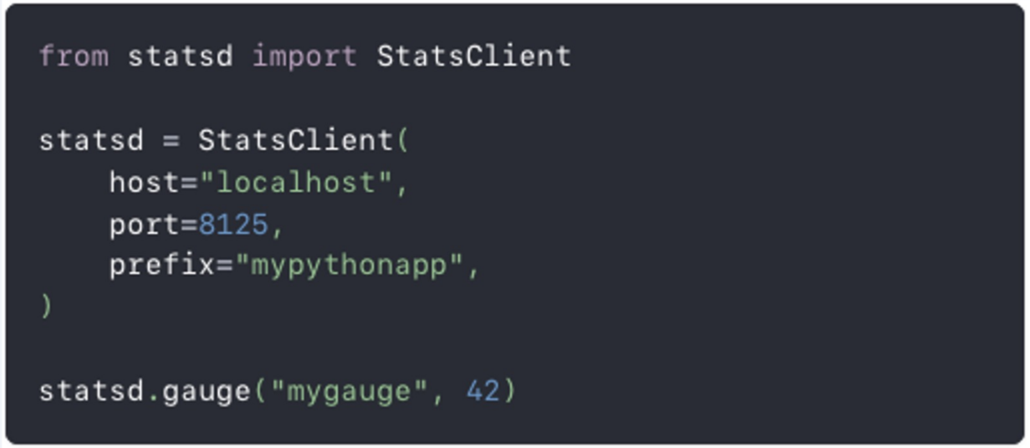
- Memfault leverages collectd to capture **system** and **device** metrics
- Customize which system metrics to capture using **collectd plugins**
- Push device metrics using the **collectd/statsd** endpoint



Pushing custom metrics



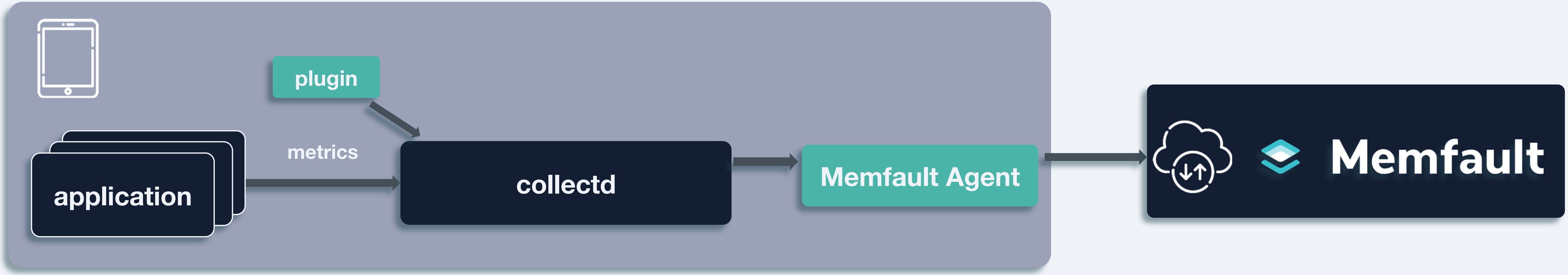
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <statsd-client.h>
#include <unistd.h>

#define MAX_LINE_LEN 200
#define PKT_LEN 1400

int main(int argc, char *argv[])
{
  statsd_link *link;

  link = statsd_init_with_namespace("localhost", 8125, "mycapp");

  char pkt[PKT_LEN] = {'\0'};
  char tmp[MAX_LINE_LEN] = {'\0'};

  statsd_prepare(link, "mygauge", 42, "g", 1.0, tmp, MAX_LINE_LEN, 1);
  strncat(pkt, tmp, PKT_LEN - 1);
  statsd_send(link, pkt);

  statsd_finalize(link);
}
```



```python
from statsd import StatsClient

statsd = StatsClient(
    host="localhost",
    port=8125,
    prefix="mypythonapp",
)

statsd.gauge("mygauge", 42)
```

Look for a statsd library in your language

Data aggregation that can scale



statsd push
Arbitrary frequency

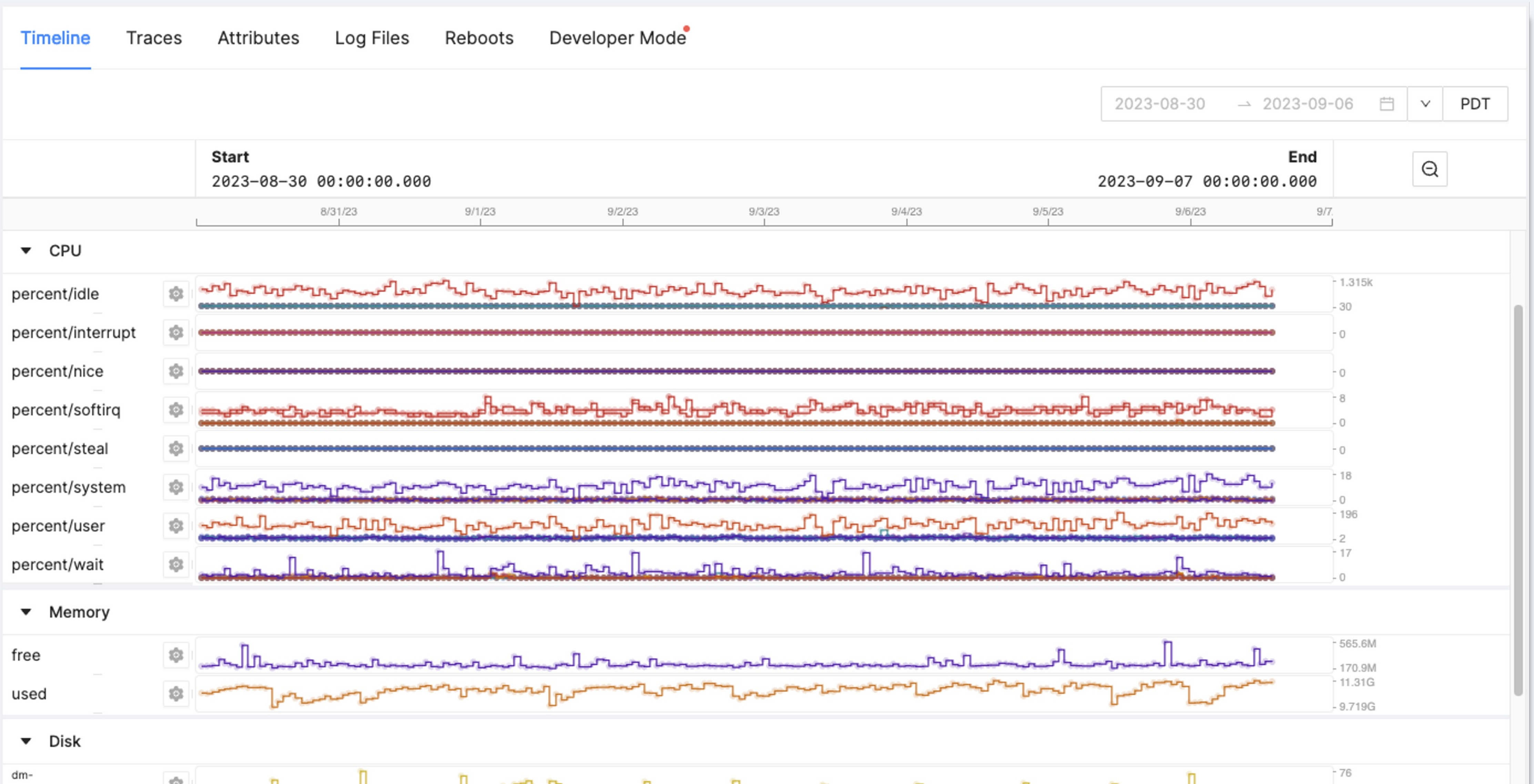
collectd plugins
Typically 10s interval

Collectd Aggregation
Push all metrics every 10s

Memfaultd Aggregation
One heartbeat /hour

Backend Aggregation
Maintain /hour and /day aggregation for all timeseries

Device monitoring

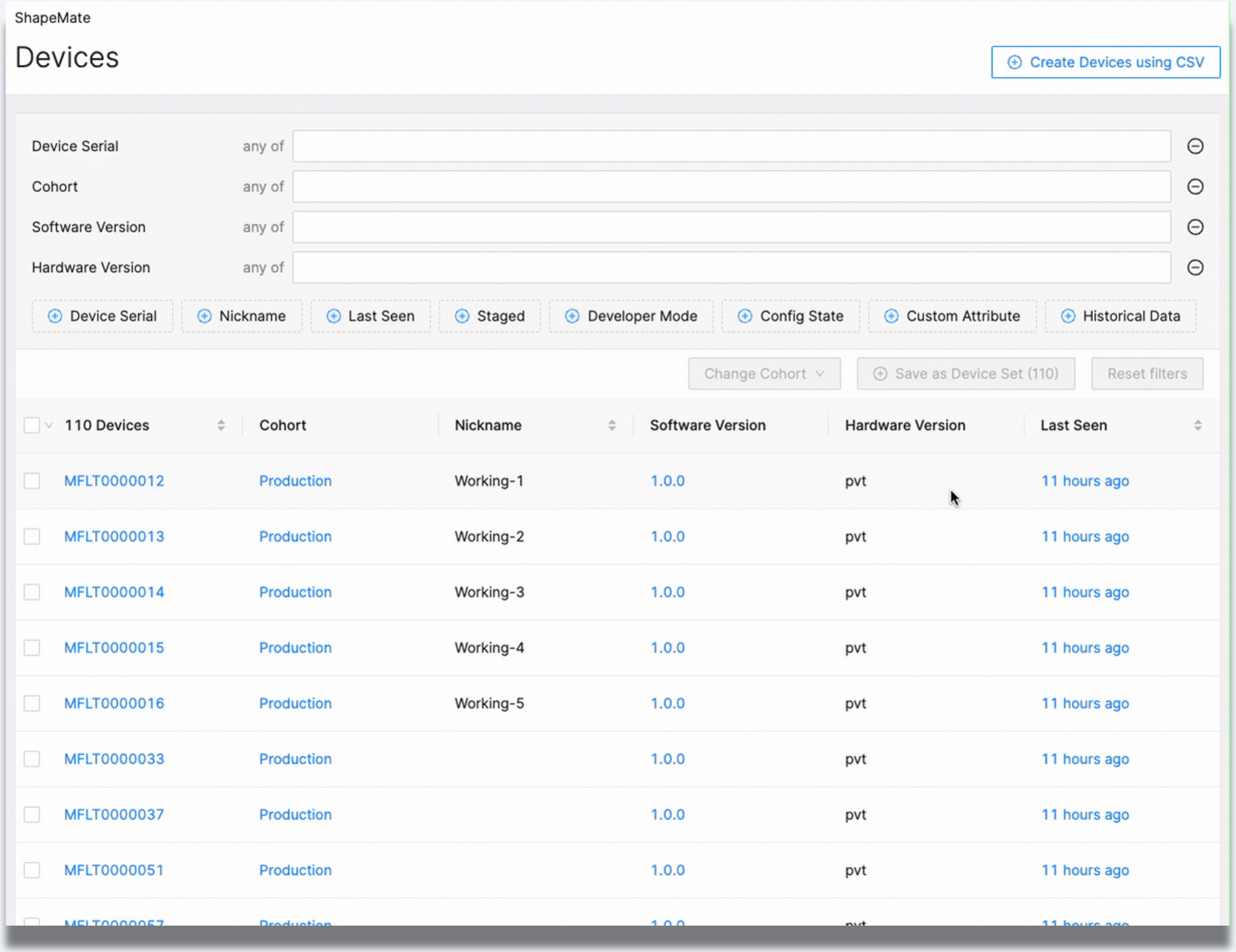


Start
2023-08-30 00:00:00.000

End
2023-09-07 00:00:00.000

Use metrics to create a device set

- Device sets are dynamic list of devices
- The list will update as new data comes in



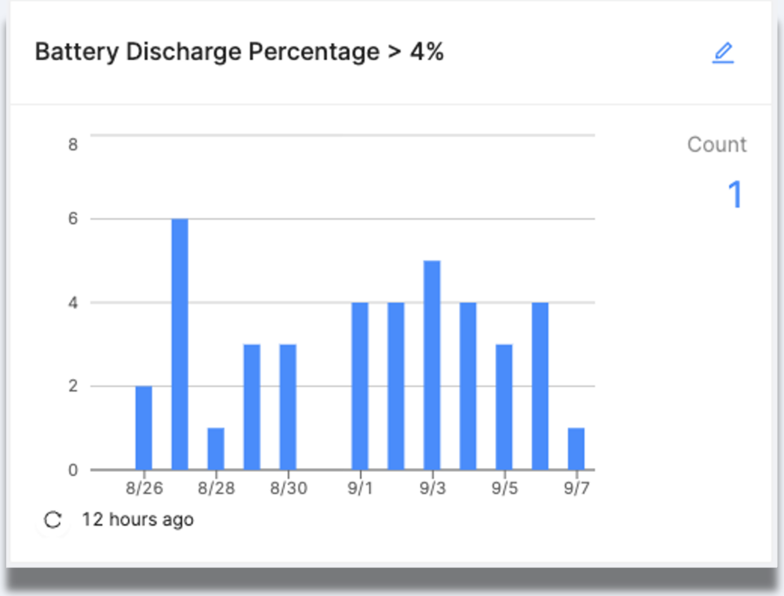
ShapeMate

Devices

Device Serial	any of	
Cohort	any of	
Software Version	any of	
Hardware Version	any of	

□	110 Devices	Cohort	Nickname	Software Version	Hardware Version	Last Seen
□	MFLT0000012	Production	Working-1	1.0.0	pvt	11 hours ago
□	MFLT0000013	Production	Working-2	1.0.0	pvt	11 hours ago
□	MFLT0000014	Production	Working-3	1.0.0	pvt	11 hours ago
□	MFLT0000015	Production	Working-4	1.0.0	pvt	11 hours ago
□	MFLT0000016	Production	Working-5	1.0.0	pvt	11 hours ago
□	MFLT0000033	Production		1.0.0	pvt	11 hours ago
□	MFLT0000037	Production		1.0.0	pvt	11 hours ago
□	MFLT0000051	Production		1.0.0	pvt	11 hours ago
□	MFLT0000057	Production		1.0.0	pvt	11 hours ago

Metrics for alerting



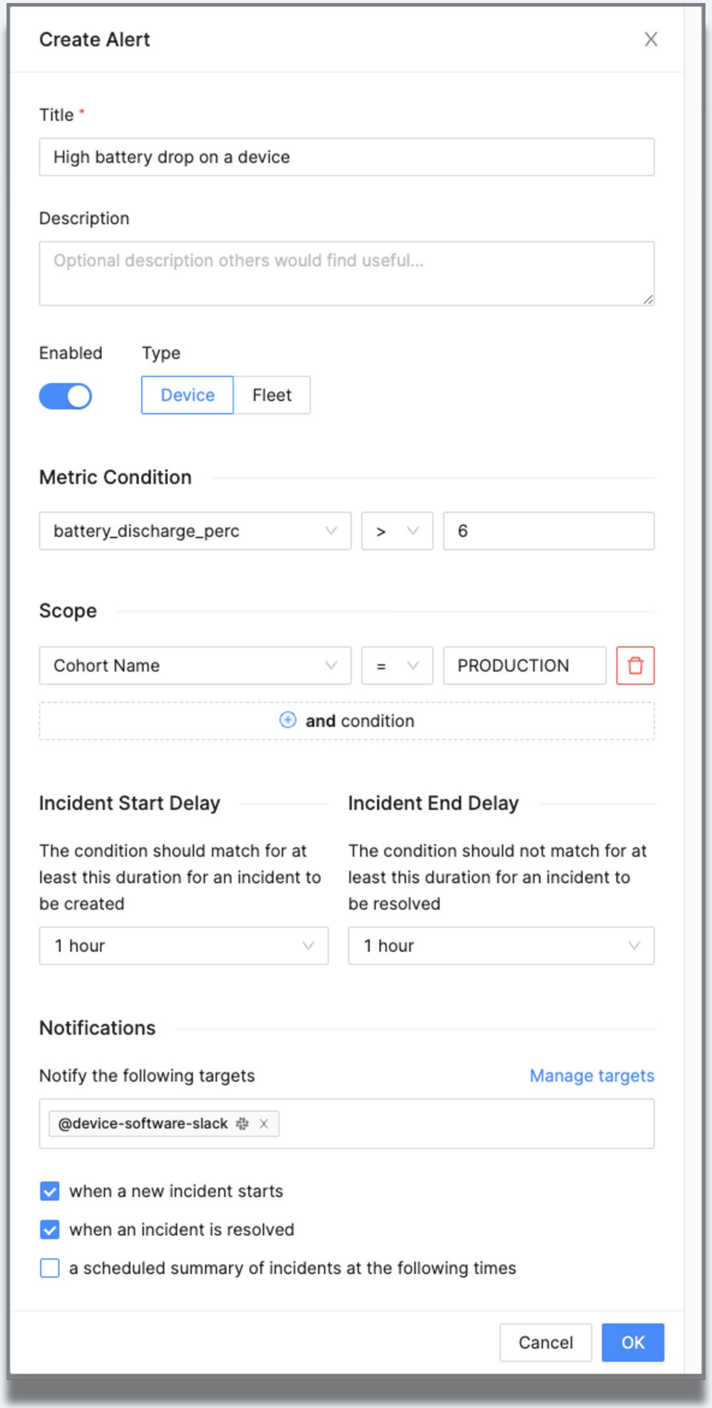
Battery Discharge Percentage > 4%

Count

1

12 hours ago

- Device sets make very useful graphs
- Metrics can be used to trigger alerts



Create Alert

Title *

High battery drop on a device

Description

Optional description others would find useful...

Enabled: ●

Type: [✓ Device] Fleet

Metric Condition

battery_discharge_perc | > | 6

Scope

Cohort Name | = | PRODUCTION

⊕ and condition

Incident Start Delay

The condition should match for at least this duration for an incident to be created

1 hour

Incident End Delay

The condition should not match for at least this duration for an incident to be resolved

1 hour

Notifications

Notify the following targets

Manage targets

@device-software-slack ×

☑ when a new incident starts

☑ when an incident is resolved

□ a scheduled summary of incidents at the following times

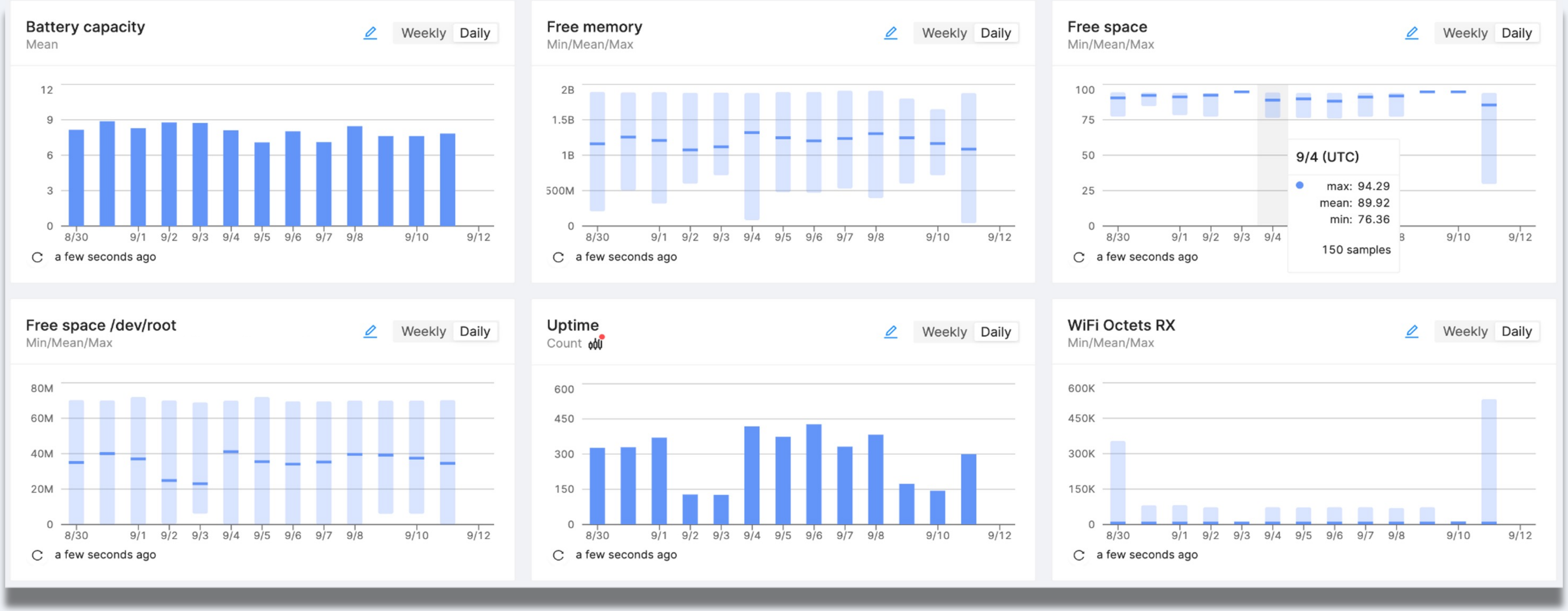
Cancel | OK

From device monitoring to fleet monitoring

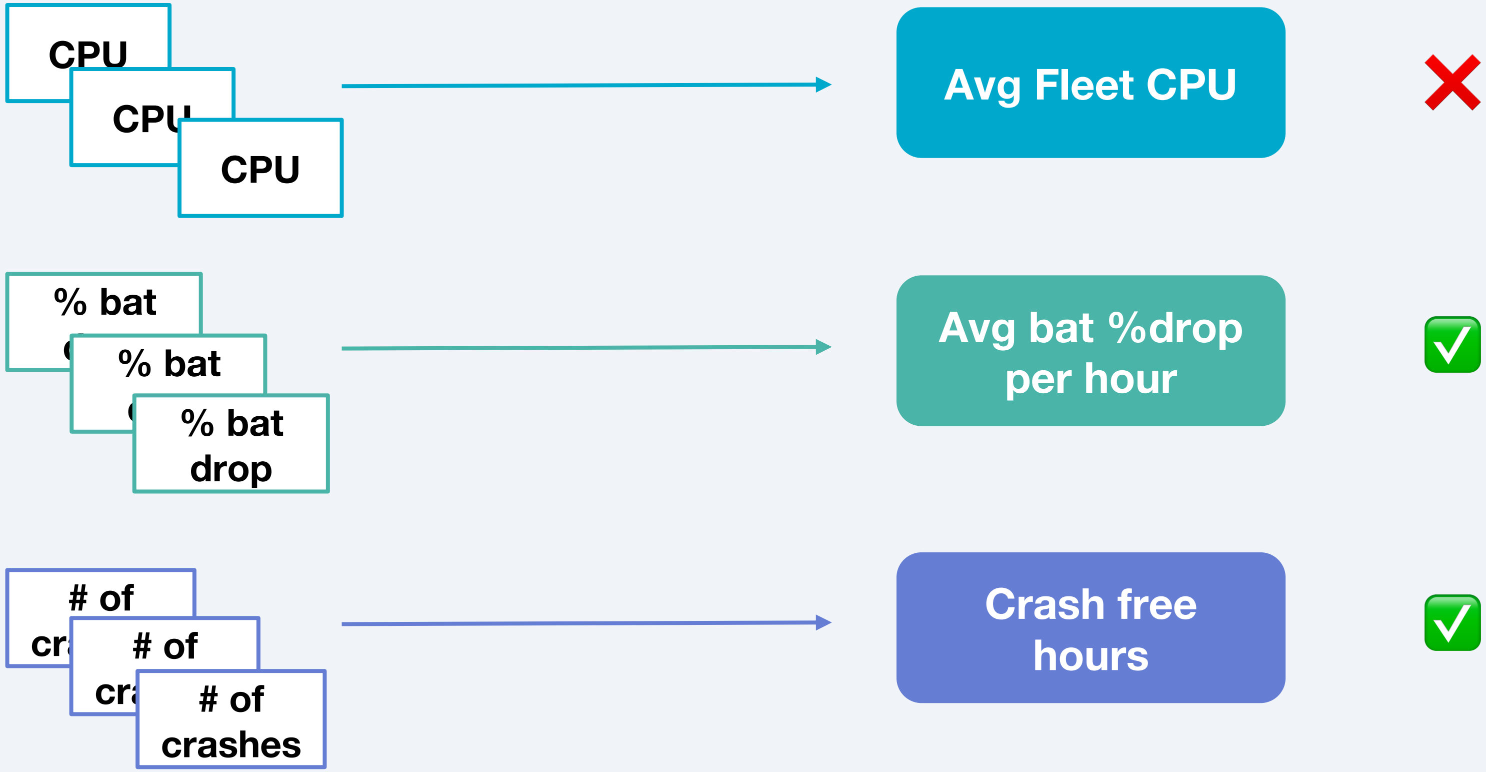
Fleet monitoring

(1000 devices or more)

- Select the right metrics
- Scale the ingestion and aggregation
- Pick the right visualization



From device to fleet monitoring



CPU / CPU / CPU → Avg Fleet CPU ❌ Not very useful unless usage is stable

% bat / % bat / % bat drop → Avg bat %drop per hour ✅ Useful to keep track of power management improvement

# of cr / # of cr / # of crashes → Crash free hours ✅ A useful KPI for device engineering teams

See detailed how to setup crash free hours
https://docs.memfault.com/docs/best-practices/fleet-reliability-metrics-crash-free-hours/

Some useful fleet metrics

Battery

- Discharge per hour (s)
- Screen “on” time (s)

Connectivity

- Cellular modem connected (s)
- Bytes received / sent
- Time spent connecting
- Ping
- RSSI

Flash

- Flash read/write
- Free disk space

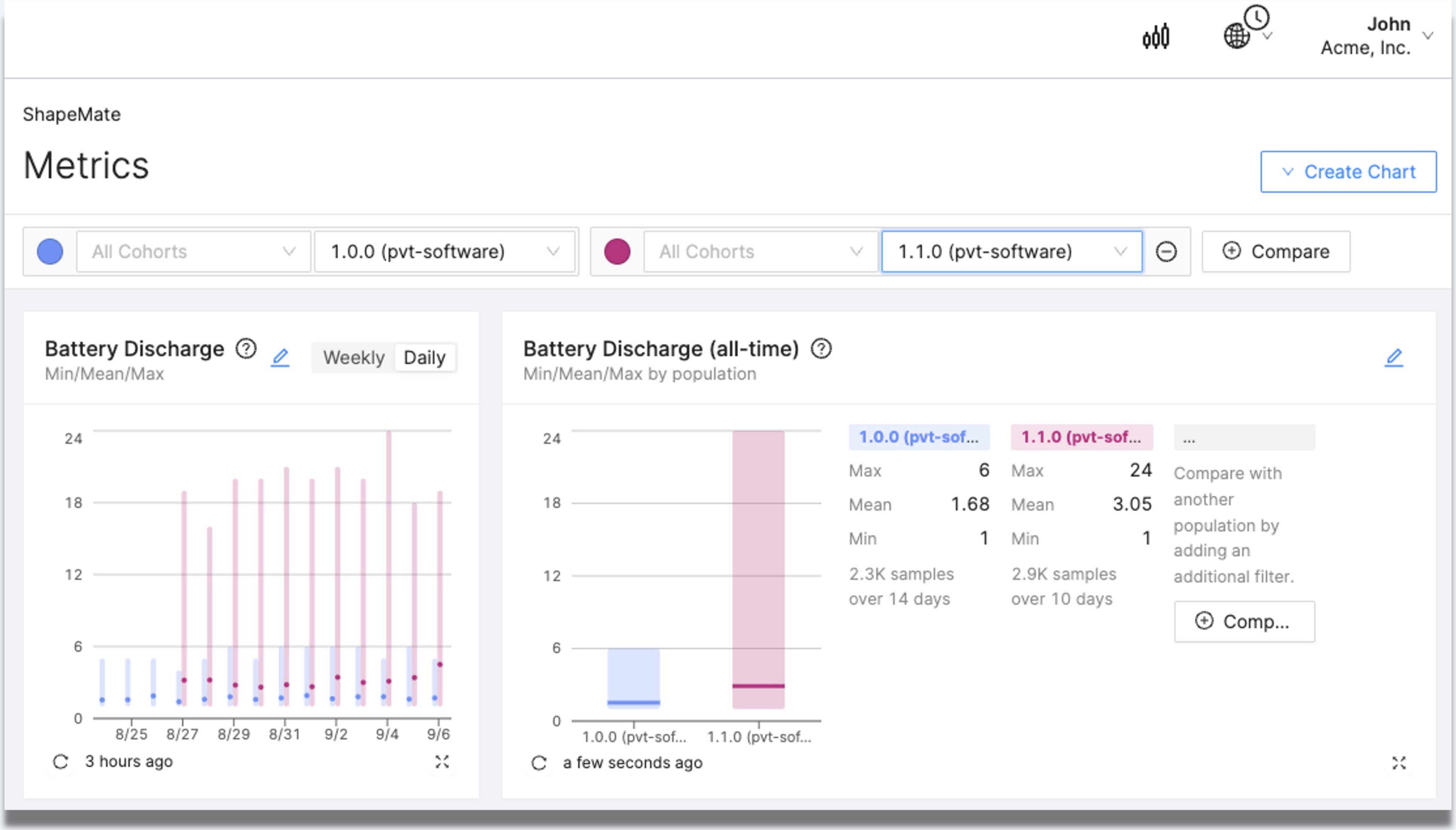
Usage

- # of operations / failure
- Time to execute operation

System

- Free memory (Min and Avg)
- Temperature

Visualizing Fleet Data



ShapeMate

Metrics

All Cohorts — 1.0.0 (pvt-software)

All Cohorts — 1.1.0 (pvt-software)

Battery Discharge

Min/Mean/Max

Battery Discharge (all-time)

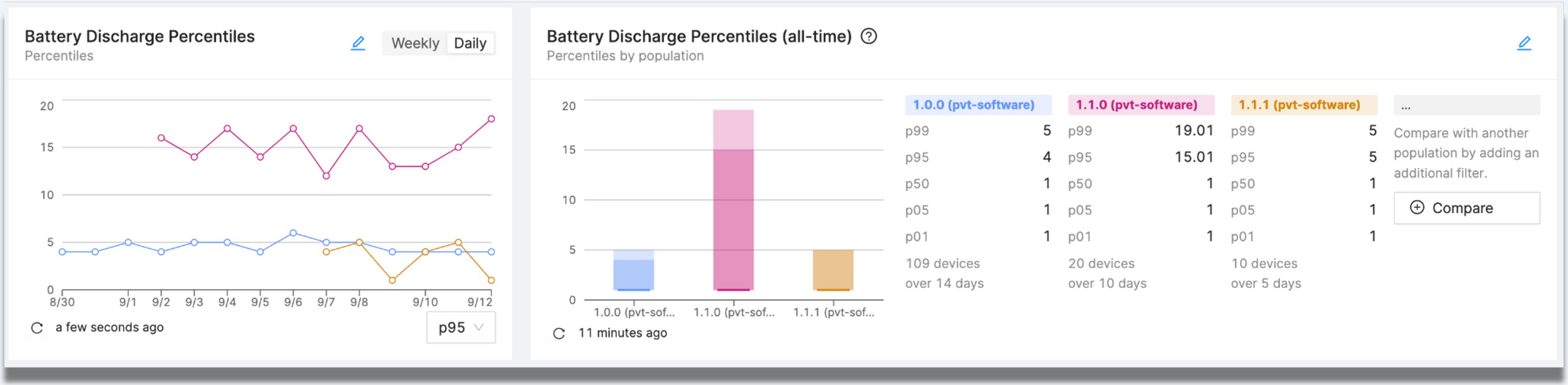
Min/Mean/Max by population

1.0.0 (pvt-sof...		1.1.0 (pvt-sof...	
Max	6	Max	24
Mean	1.68	Mean	3.05
Min	1	Min	1
2.3K samples over 14 days		2.9K samples over 10 days	

...

Compare with another population by adding an additional filter.

Visualizing Fleet Data



Battery Discharge Percentiles

Percentiles

a few seconds ago

p95

Battery Discharge Percentiles (all-time)

Percentiles by population

11 minutes ago

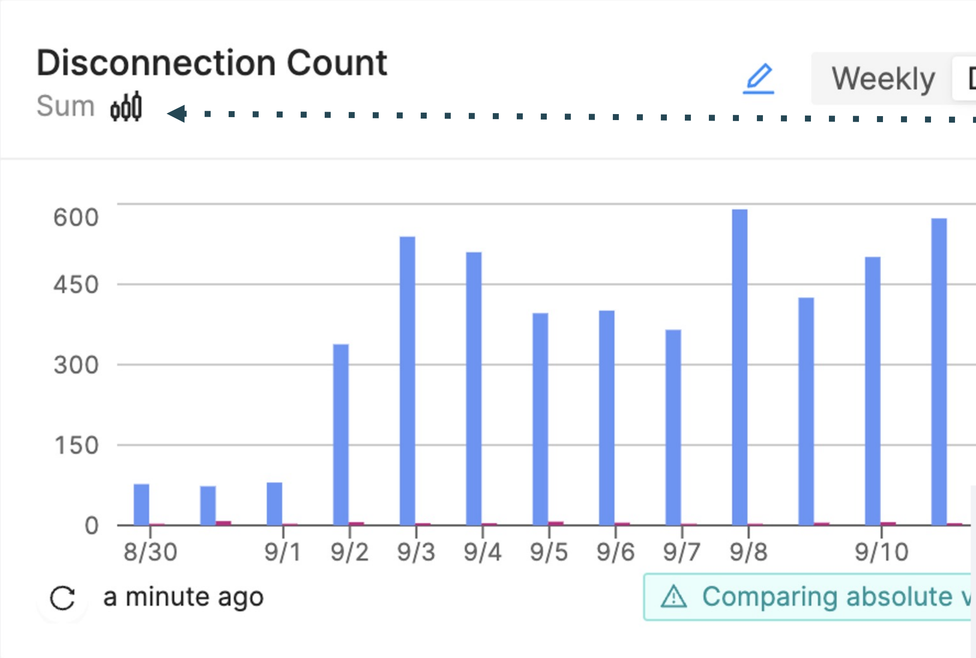
1.0.0 (pvt-software)		1.1.0 (pvt-software)		1.1.1 (pvt-software)		...
p99	5	p99	19.01	p99	5	
p95	4	p95	15.01	p95	5	
p50	1	p50	1	p50	1	
p05	1	p05	1	p05	1	
p01	1	p01	1	p01	1	
109 devices over 14 days		20 devices over 10 days		10 devices over 5 days		

Compare with another population by adding an additional filter.

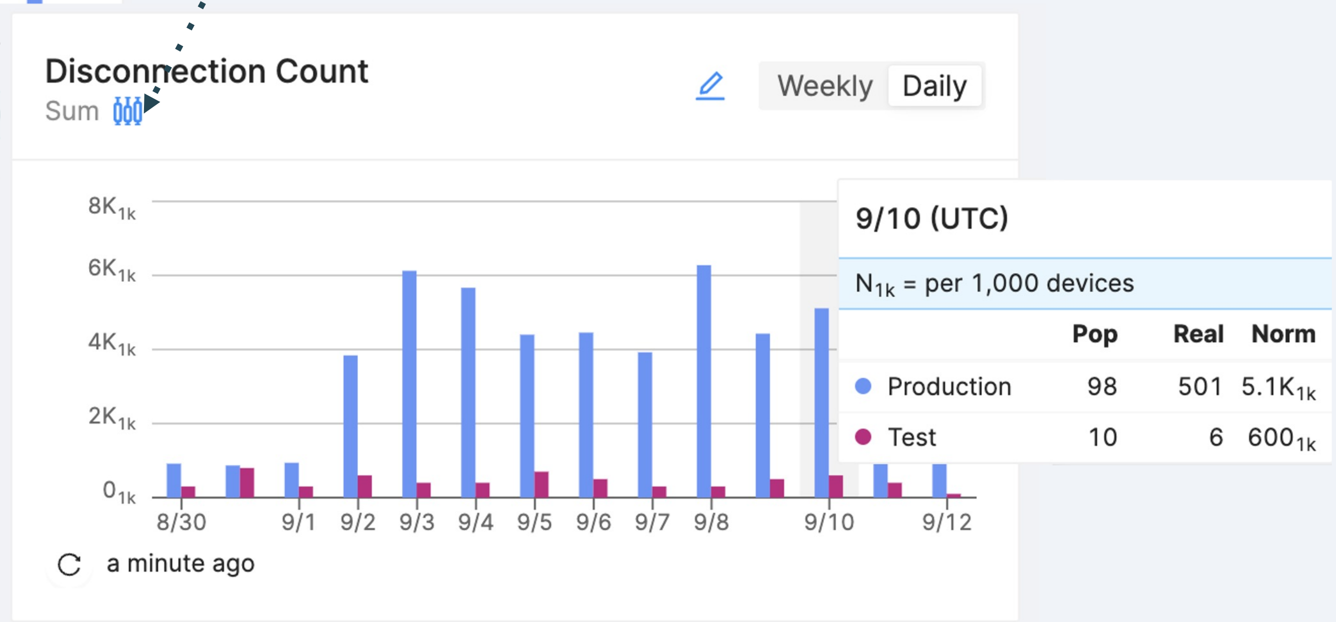
⊕ Compare

Percentiles help understand how large a problem is and how much of the fleet is impacted.

Normalizing data



Turn on data normalization to compare absolute values across cohorts of different sizes



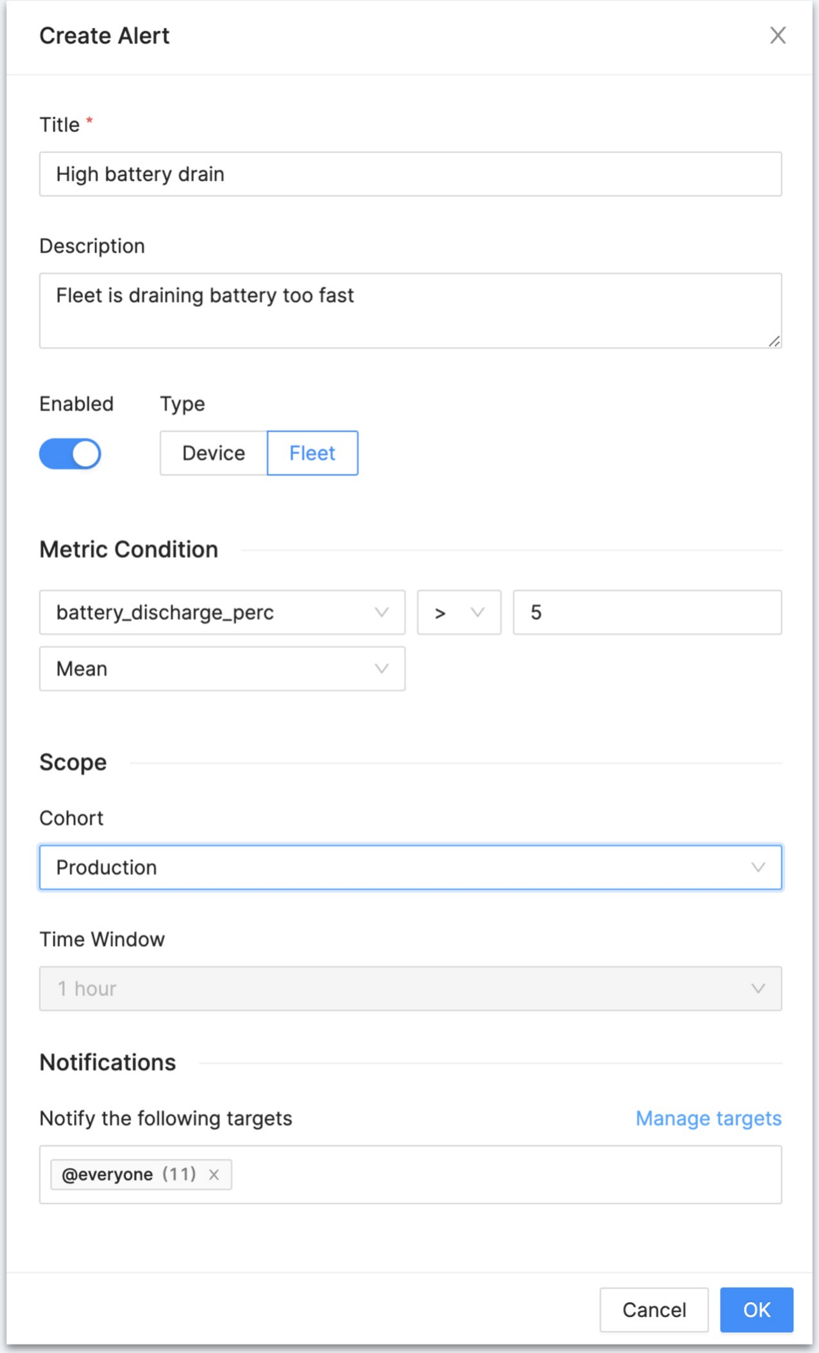
9/10 (UTC)

N_{1k} = per 1,000 devices

	Pop	Real	Norm
● Production	98	501	$5.1K_{1k}$
● Test	10	6	600_{1k}

Fleet wide alerts

- Use “Fleet Alerts” to monitor a specific metric over the entire fleet
- Send notification by email and slack to the team



Create Alert

Title *

High battery drain

Description

Fleet is draining battery too fast

Enabled

Type

Device | Fleet

Metric Condition

battery_discharge_perc > 5

Mean

Scope

Cohort

Production

Time Window

1 hour

Notifications

Notify the following targets

@everyone (11)

Monitoring Challenges

On Device

- Collecting from different sources and languages
- Partial connectivity
- Flash wear and networking costs

Easy to use "fire and forget" metrics API
On device buffering and aggregation

Backend

- Scaling pains
- Lack of flexibility
- Visualization tools

Memfault

Usage

- Drowning in data
- Metrics are meaningless when aggregated
- Signal lost in the data

Use best practices to select useful variables and iterate
Use normalization, percentiles, etc

Debugging devices

Poll #2

Where are most of your bugs?

A. Kernel

B. Kernel Modules / Drivers

C. System Daemons

D. Libraries and Runtimes

E. Application Code

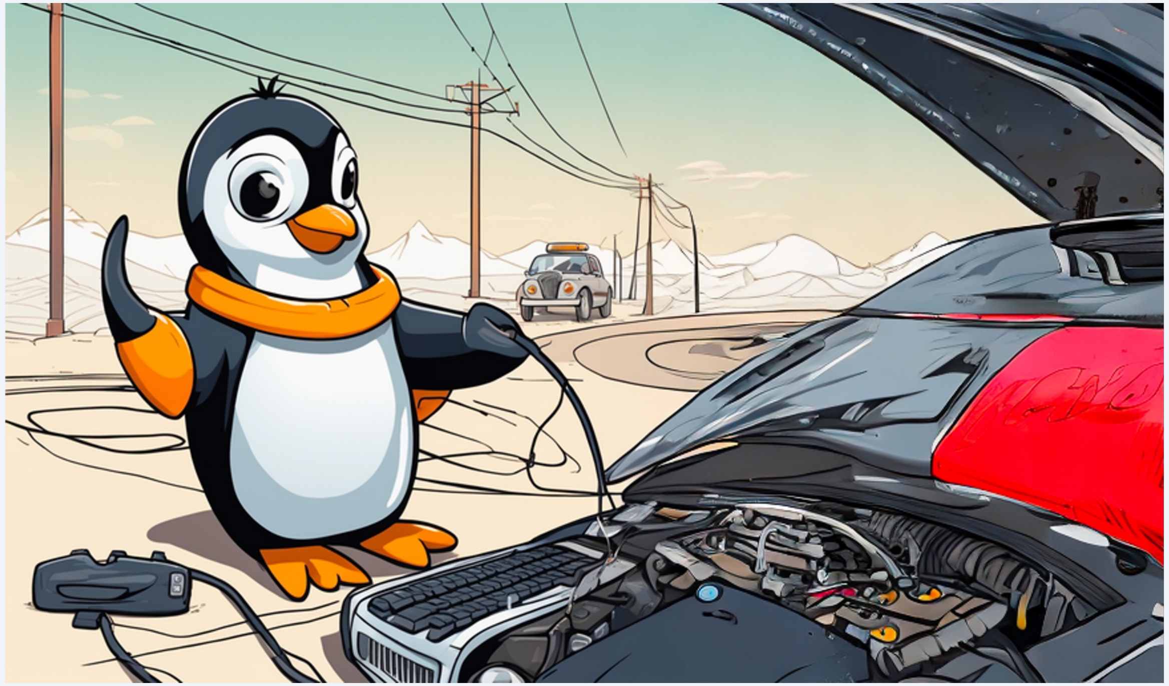
Challenges

Poor quality bug reports

Triaging bug reports

No access to the device

No visibility inside the device



Debugging Tools

Device Metrics

Logs

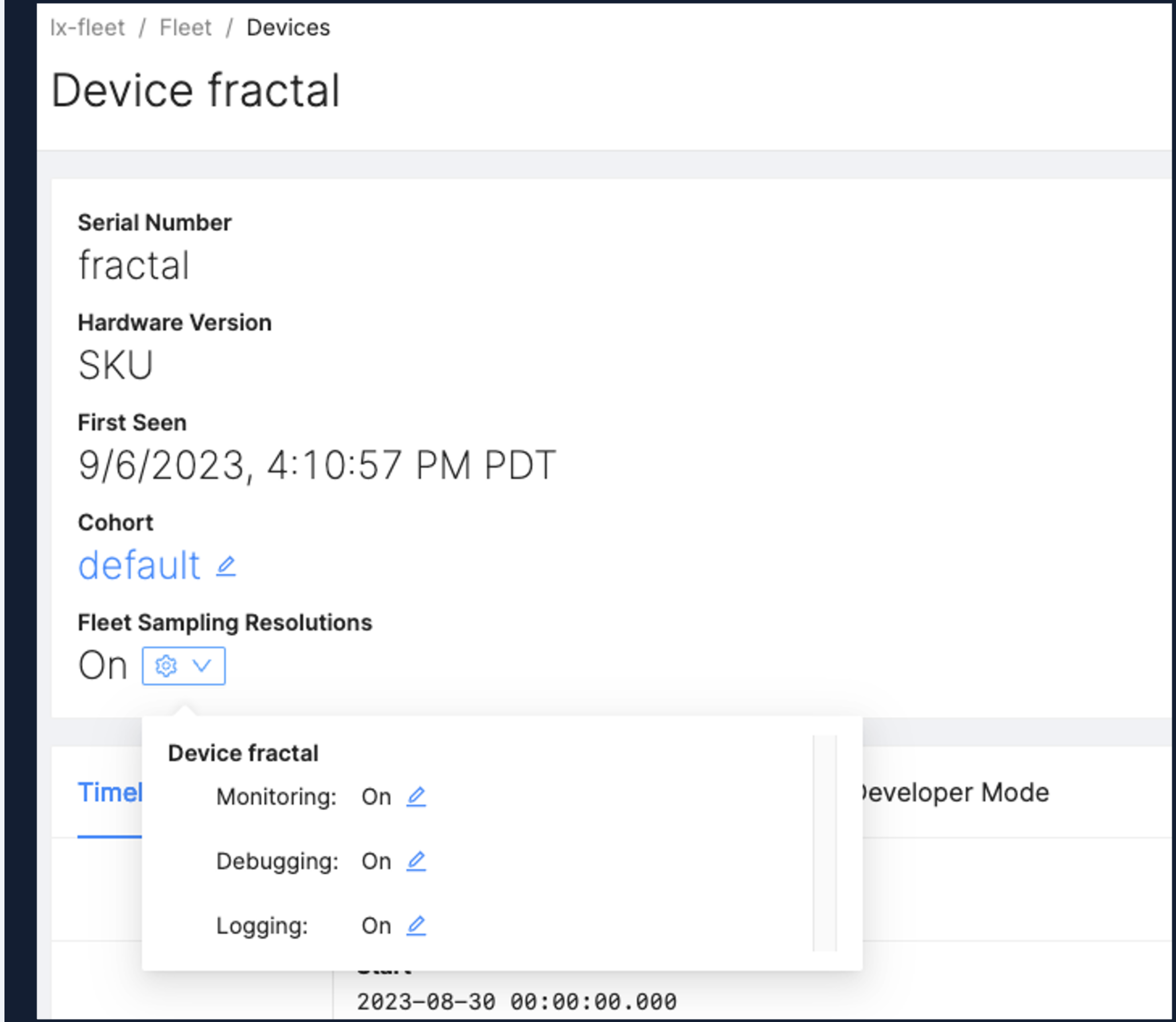
Coredumps

Logs

Memfault captures device logs with fluent-bit and stores them locally on device.

By default logs are only uploaded for development devices.

You can selectively collect logs from specific devices. This works retroactively.



lx-fleet / Fleet / Devices

Device fractal

Serial Number
fractal

Hardware Version
SKU

First Seen
9/6/2023, 4:10:57 PM PDT

Cohort
default

Fleet Sampling Resolutions
On

Device fractal

Monitoring: On

Debugging: On

Logging: On

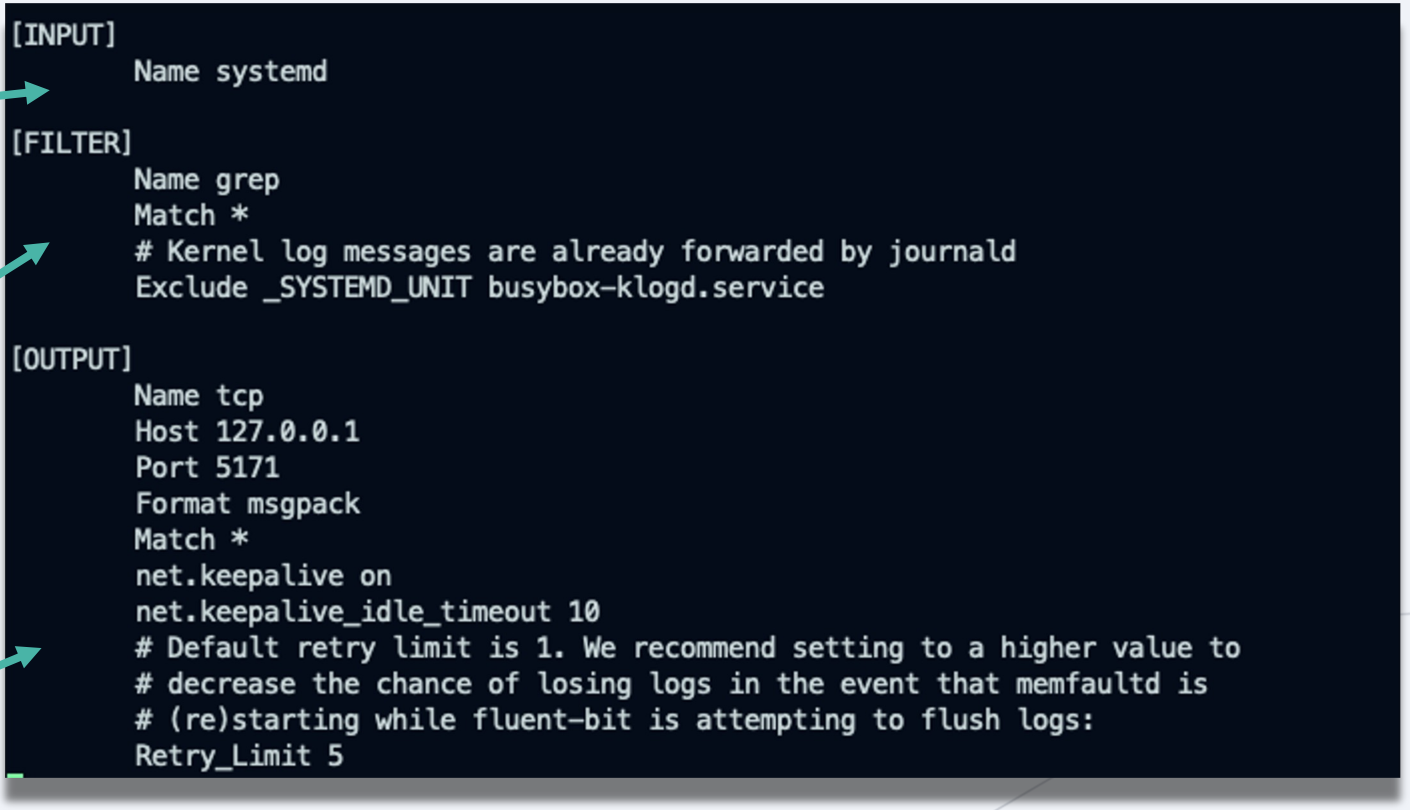
2023-08-30 00:00:00.000

Configuring fluent-bit

Many input options: systemd, files, network, serial

On “the edge” filtering to reduce noise

fluent-bit does not write to disk Sends to memfaultd

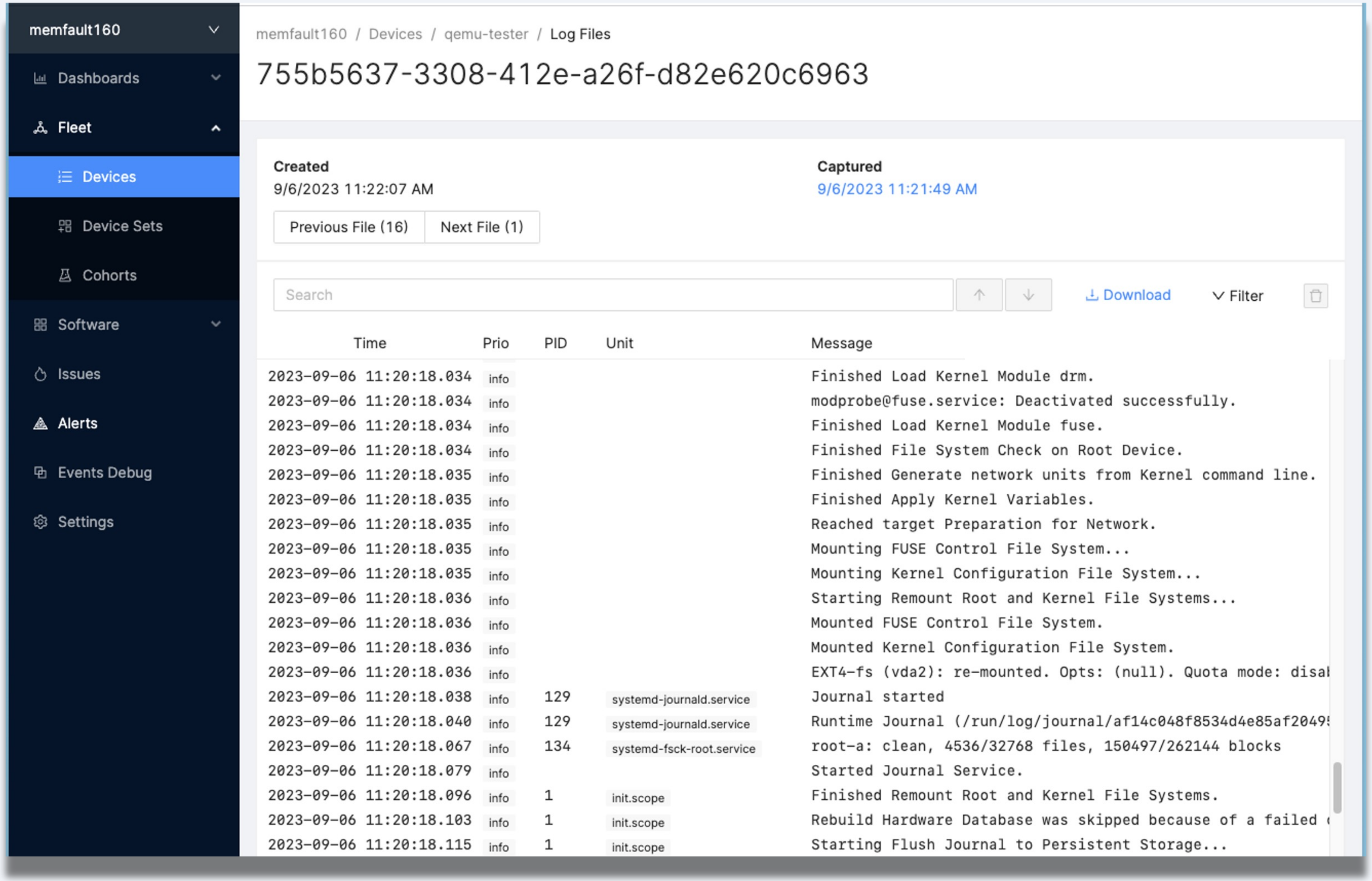


```
[INPUT]
        Name systemd

[FILTER]
        Name grep
        Match *
        # Kernel log messages are already forwarded by journald
        Exclude _SYSTEMD_UNIT busybox-klogd.service

[OUTPUT]
        Name tcp
        Host 127.0.0.1
        Port 5171
        Format msgpack
        Match *
        net.keepalive on
        net.keepalive_idle_timeout 10
        # Default retry limit is 1. We recommend setting to a higher value to
        # decrease the chance of losing logs in the event that memfaultd is
        # (re)starting while fluent-bit is attempting to flush logs:
        Retry_Limit 5
```

memfaultd applies rate limiting, compresses and writes to disk at regular intervals (10MB or 1hour)

Logs are kept on disk until requested by backend or max storage space is exhausted.

Browsing logs



755b5637-3308-412e-a26f-d82e620c6963

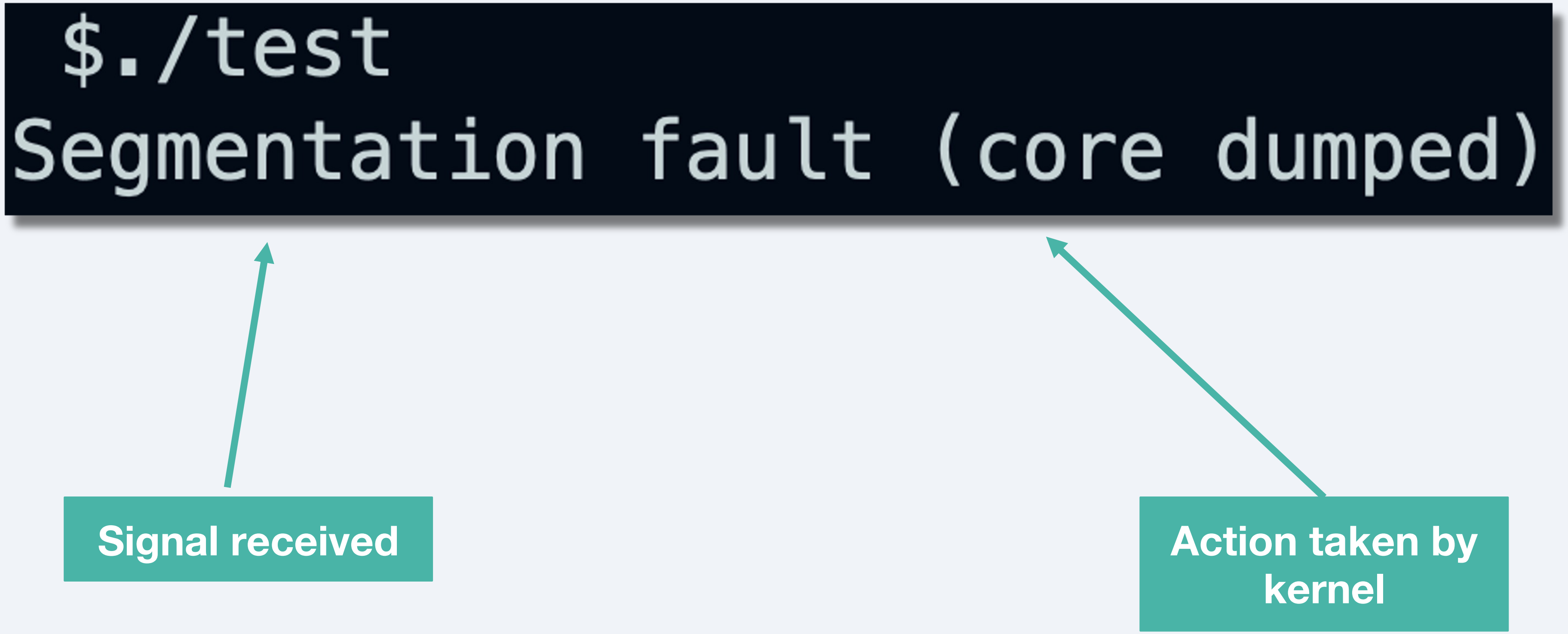
Created
9/6/2023 11:22:07 AM

Captured
9/6/2023 11:21:49 AM

```
          Time          Prio   PID    Unit                         Message
2023-09-06 11:20:18.034  info                                      Finished Load Kernel Module drm.
2023-09-06 11:20:18.034  info                                      modprobe@fuse.service: Deactivated successfully.
2023-09-06 11:20:18.034  info                                      Finished Load Kernel Module fuse.
2023-09-06 11:20:18.034  info                                      Finished File System Check on Root Device.
2023-09-06 11:20:18.035  info                                      Finished Generate network units from Kernel command line.
2023-09-06 11:20:18.035  info                                      Finished Apply Kernel Variables.
2023-09-06 11:20:18.035  info                                      Reached target Preparation for Network.
2023-09-06 11:20:18.035  info                                      Mounting FUSE Control File System...
2023-09-06 11:20:18.035  info                                      Mounting Kernel Configuration File System...
2023-09-06 11:20:18.036  info                                      Starting Remount Root and Kernel File Systems...
2023-09-06 11:20:18.036  info                                      Mounted FUSE Control File System.
2023-09-06 11:20:18.036  info                                      Mounted Kernel Configuration File System.
2023-09-06 11:20:18.036  info                                      EXT4-fs (vda2): re-mounted. Opts: (null). Quota mode: disa
2023-09-06 11:20:18.038  info  129    systemd-journald.service     Journal started
2023-09-06 11:20:18.040  info  129    systemd-journald.service     Runtime Journal (/run/log/journal/af14c048f8534d4e85af2049
2023-09-06 11:20:18.067  info  134    systemd-fsck-root.service    root-a: clean, 4536/32768 files, 150497/262144 blocks
2023-09-06 11:20:18.079  info                                      Started Journal Service.
2023-09-06 11:20:18.096  info  1      init.scope                   Finished Remount Root and Kernel File Systems.
2023-09-06 11:20:18.103  info  1      init.scope                   Rebuild Hardware Database was skipped because of a failed
2023-09-06 11:20:18.115  info  1      init.scope                   Starting Flush Journal to Persistent Storage...
```

Logs are also available via an API endpoint

Debugging with coredumps

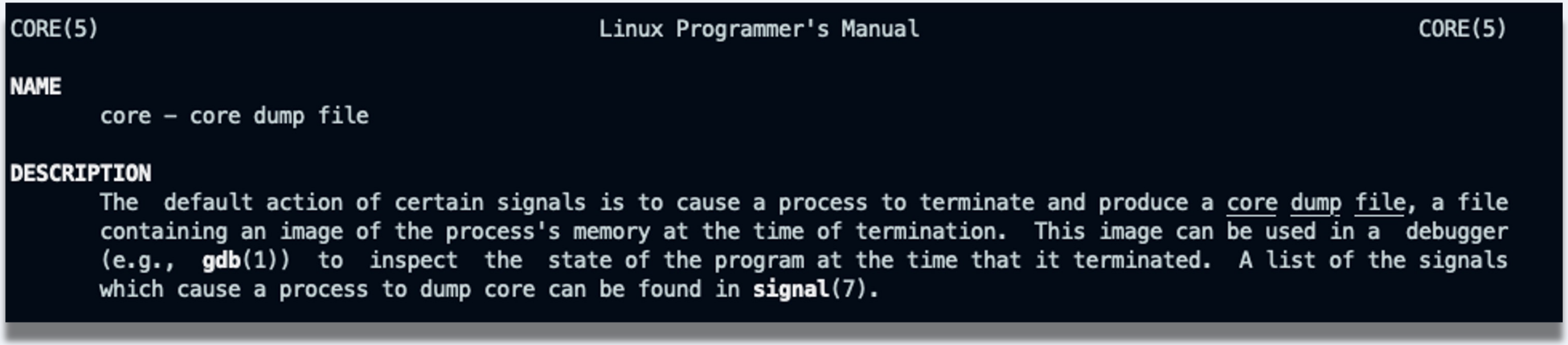


```
 $./test
Segmentation fault (core dumped)
```

Signal received

Action taken by kernel

Coredump

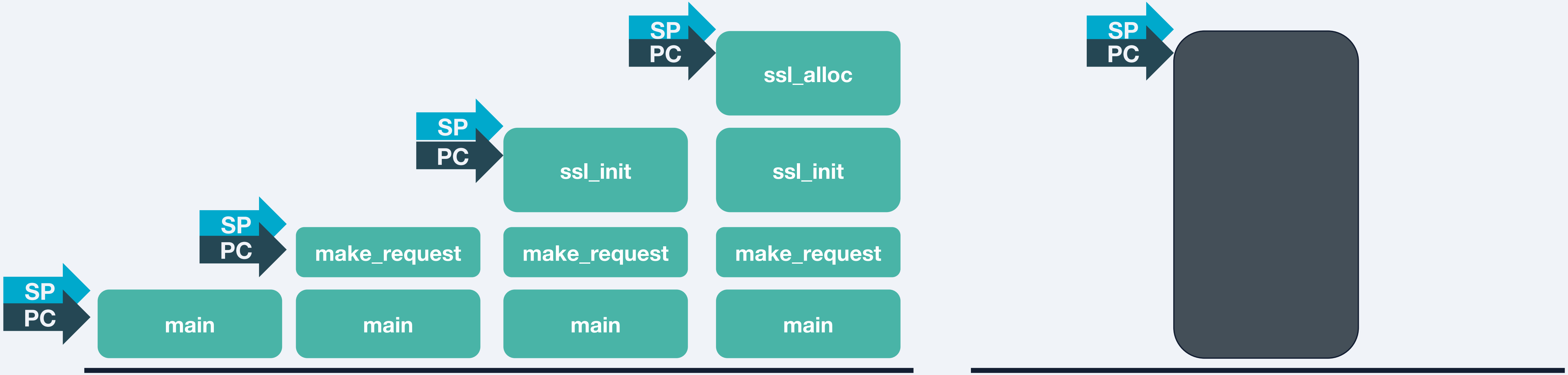


```
CORE(5)                          Linux Programmer's Manual                          CORE(5)

NAME
       core - core dump file

DESCRIPTION
       The  default action of certain signals is to cause a process to terminate and produce a core dump file, a file
       containing an image of the process's memory at the time of termination.  This image can be used in a  debugger
       (e.g.,  gdb(1))  to  inspect  the  state of the program at the time that it terminated.  A list of the signals
       which cause a process to dump core can be found in signal(7).
```

Coredumps contain:

- program status for each thread (registers incl. PC and SP)
- some of the program memory (mostly stack and heap)
- build-id of the running binary
- build-id and address of all the dynamic libraries that are loaded

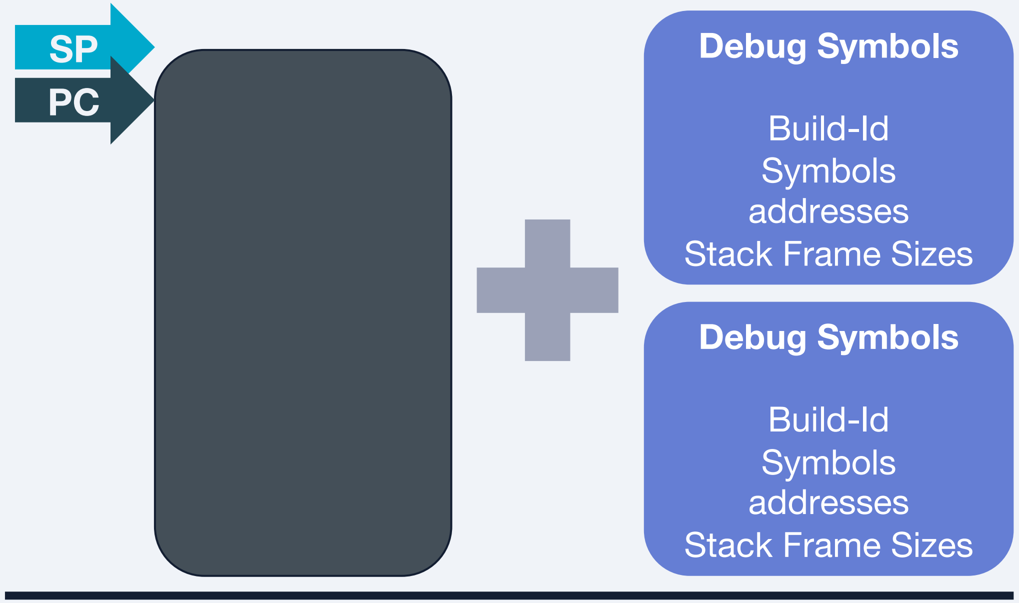
Using coredumps



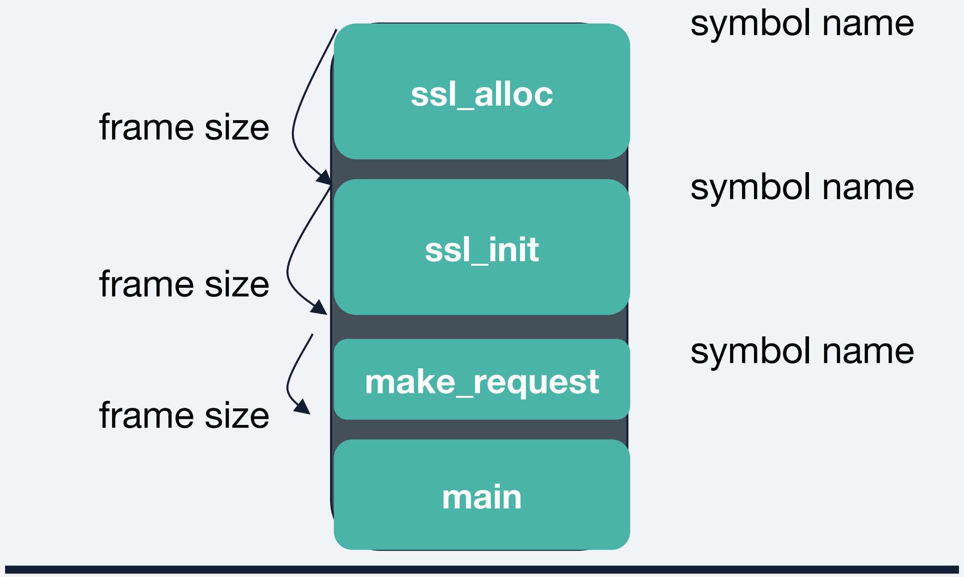
what happens on the device

data we get
in the coredump

Using coredumps



what we get



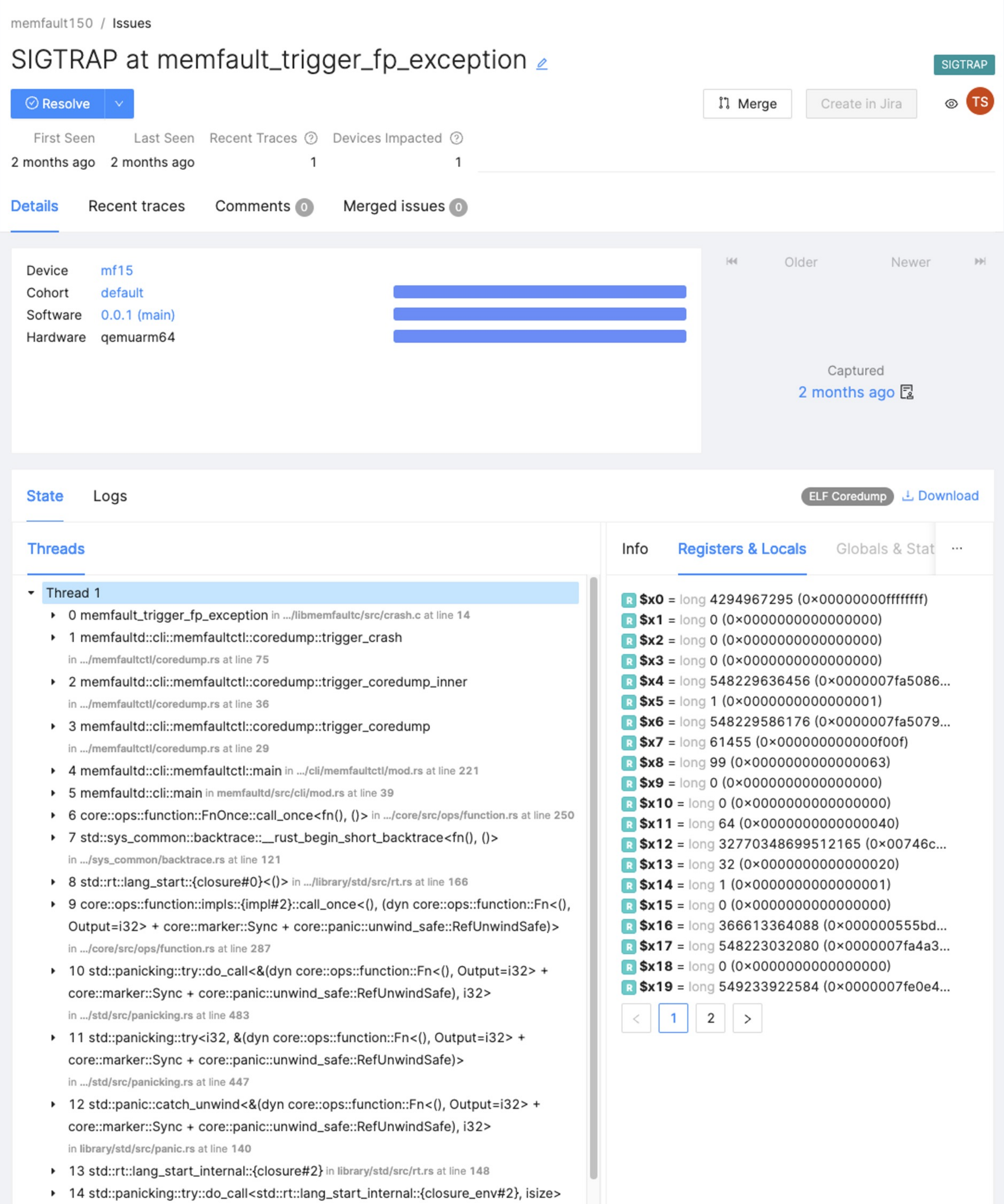
what we can recover

Coredumps view

The state view will list all the running threads and their status

Typically, you will get an immediate read on where the error happened and what were the local variables at the crash

For more complicated bugs, you can download the coredump and run the debugger locally



SIGTRAP at memfault_trigger_fp_exception

First Seen	Last Seen	Recent Traces	Devices Impacted
2 months ago	2 months ago	1	1

Device mf15
Cohort default
Software 0.0.1 (main)
Hardware qemuarm64

Captured
2 months ago

```
Thread 1
  0 memfault_trigger_fp_exception in .../libmemfaultc/src/crash.c at line 14
  1 memfaultd::cli::memfaultctl::coredump::trigger_crash
    in .../memfaultctl/coredump.rs at line 75
  2 memfaultd::cli::memfaultctl::coredump::trigger_coredump_inner
    in .../memfaultctl/coredump.rs at line 36
  3 memfaultd::cli::memfaultctl::coredump::trigger_coredump
    in .../memfaultctl/coredump.rs at line 29
  4 memfaultd::cli::memfaultctl::main in .../cli/memfaultctl/mod.rs at line 221
  5 memfaultd::cli::main in memfaultd/src/cli/mod.rs at line 39
  6 core::ops::function::FnOnce::call_once<fn(), ()> in .../core/src/ops/function.rs at line 250
  7 std::sys_common::backtrace::__rust_begin_short_backtrace<fn(), ()>
    in .../sys_common/backtrace.rs at line 121
  8 std::rt::lang_start::{closure#0}<()> in .../library/std/src/rt.rs at line 166
  9 core::ops::function::impls::{impl#2}::call_once<(), (dyn core::ops::function::Fn<(),
    Output=i32> + core::marker::Sync + core::panic::unwind_safe::RefUnwindSafe)>
    in .../core/src/ops/function.rs at line 287
  10 std::panicking::try::do_call<&(dyn core::ops::function::Fn<(), Output=i32> +
    core::marker::Sync + core::panic::unwind_safe::RefUnwindSafe), i32>
    in .../std/src/panicking.rs at line 483
  11 std::panicking::try<i32, &(dyn core::ops::function::Fn<(), Output=i32> +
    core::marker::Sync + core::panic::unwind_safe::RefUnwindSafe)>
    in .../std/src/panicking.rs at line 447
  12 std::panic::catch_unwind<&(dyn core::ops::function::Fn<(), Output=i32> +
    core::marker::Sync + core::panic::unwind_safe::RefUnwindSafe), i32>
    in library/std/src/panic.rs at line 140
  13 std::rt::lang_start_internal::{closure#2} in library/std/src/rt.rs at line 148
  14 std::panicking::try::do_call<std::rt::lang_start_internal::{closure_env#2}, isize>
```

```
R $x0 = long 4294967295 (0×00000000ffffffff)
R $x1 = long 0 (0×0000000000000000)
R $x2 = long 0 (0×0000000000000000)
R $x3 = long 0 (0×0000000000000000)
R $x4 = long 548229636456 (0×0000007fa5086...
R $x5 = long 1 (0×0000000000000001)
R $x6 = long 548229586176 (0×0000007fa5079...
R $x7 = long 61455 (0×000000000000f00f)
R $x8 = long 99 (0×0000000000000063)
R $x9 = long 0 (0×0000000000000000)
R $x10 = long 0 (0×0000000000000000)
R $x11 = long 64 (0×0000000000000040)
R $x12 = long 32770348699512165 (0×00746c...
R $x13 = long 32 (0×0000000000000020)
R $x14 = long 1 (0×0000000000000001)
R $x15 = long 0 (0×0000000000000000)
R $x16 = long 366613364088 (0×000000555bd...
R $x17 = long 548223032080 (0×0000007fa4a3...
R $x18 = long 0 (0×0000000000000000)
R $x19 = long 549233922584 (0×0000007fe0e4...
```

< 1 2 >

Uploading Symbols

Debugging symbols are required to provide useful coredump analysis

Build and save debugging symbols for all the binaries you produce

Incuding all system libraries

Strip your binaries before sending them to customers

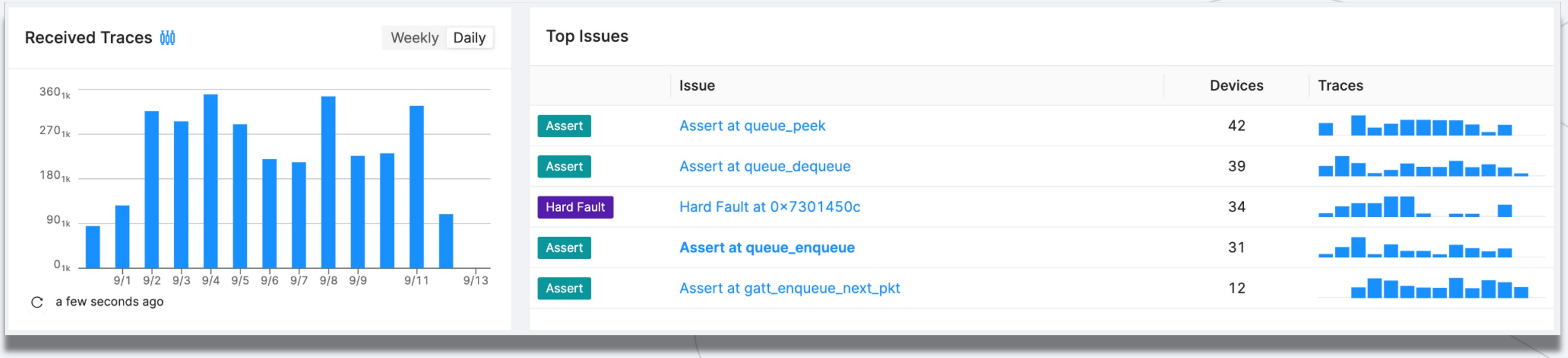
```
# Manual symbols upload

$ gcc -g -o code code.c
$ memfault upload-symbols code
$ strip code
$ cp code /mydevice/usr/bin/


# With Yocto

$ cat >> conf/local.conf
DEPENDS:append = " elfutils-native"
IMAGE_GEN_DEBUGFS = "1"
IMAGE_FSTYPES_DEBUGFS = "tar.bz2"

$ bitbake image
$ memfault upload-yocto-symbols ...
```

Using Coredumps at Scale

Memfault will automatically generate a signature for each **Trace** and group all similar traces together in one **Issue**

Keep track of

- Number of traces captured per day
- Number of devices impacted by a specific issue
- Frequency of an issue over different firmware versions



Top Issues

	Issue	Devices	Traces
Assert	Assert at queue_peek	42	
Assert	Assert at queue_dequeue	39	
Hard Fault	Hard Fault at 0×7301450c	34	
Assert	**Assert at queue_enqueue**	31	
Assert	Assert at gatt_enqueue_next_pkt	12	

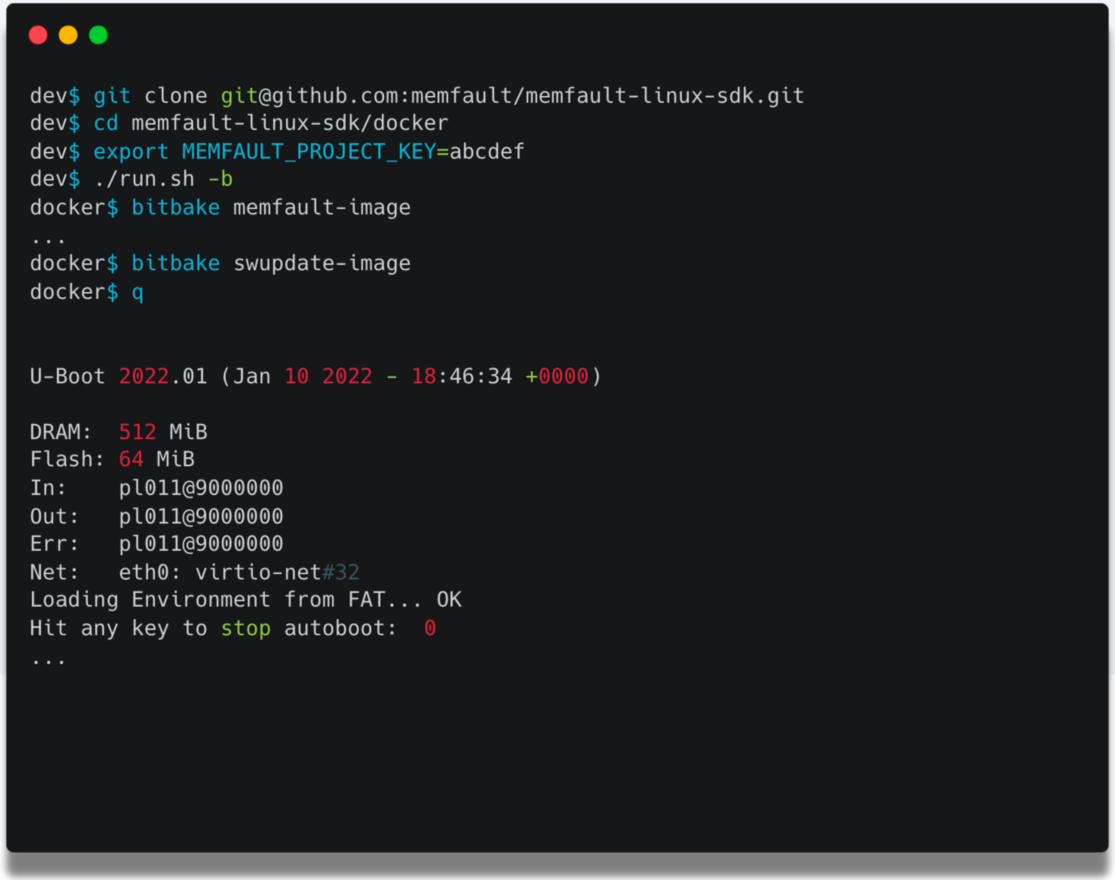
Getting started

Try this at home!

https://docs.memfault.com/docs/linux/quickstart

Memfault Linux SDK

- Docker container to easily build Yocto images
- Pre-configured for OTA with SWUpdate and U-Boot
- Runs inside QEMU or on RasperryPis



```
dev$ git clone git@github.com:memfault/memfault-linux-sdk.git
dev$ cd memfault-linux-sdk/docker
dev$ export MEMFAULT_PROJECT_KEY=abcdef
dev$ ./run.sh -b
docker$ bitbake memfault-image
...
docker$ bitbake swupdate-image
docker$ q


U-Boot 2022.01 (Jan 10 2022 - 18:46:34 +0000)

DRAM:  512 MiB
Flash: 64 MiB
In:    pl011@9000000
Out:   pl011@9000000
Err:   pl011@9000000
Net:   eth0: virtio-net#32
Loading Environment from FAT... OK
Hit any key to stop autoboot:  0
...
```

Thank You!

- memfault.com
- twitter.com/memfault
- interrupt-slack.herokuapp.com

- We’re hiring!

Memfault